Sains Malaysiana 39(6)(2010): 1041–1048

Improving Pipelined Time Stepping Algorithm for Distributed Memory Multicomputers

(Menambahbaik Algoritma Sela-Masa Penalian Paip bagi Multikomputer Memori Teragih)

NG KOK FU* & NORHASHIDAH HJ. MOHD ALI

ABSTRACT

Time stepping algorithm with spatial parallelisation is commonly used to solve time dependent partial differential equations. Computation in each time step is carried out using all processors available before sequentially advancing to the next time step. In cases where few spatial components are involved and there are relatively many processors available for use, this will result in fine granularity and decreased scalability. Naturally one alternative is to parallelise the temporal domain. Several time parallelisation algorithms have been suggested for the past two decades. One of them is the pipelined iterations across time steps. In this pipelined time stepping method, communication however is extensive between time steps during the pipelining process. This causes a decrease in performance on distributed memory environment which often has high message latency. We present a modified pipelined time stepping algorithm based on delayed pipelining and reduced communication strategies to improve overall execution time on a distributed memory environment using MPI. Our goal is to reduce the inter-time step communications while providing adequate information for the next time step to converge. Numerical result confirms that the improved algorithm is faster than the original pipelined algorithm and sequential time stepping algorithm with spatial parallelisation alone. The improved algorithm is most beneficial for fine granularity time dependent problems with limited spatial parallelisation.

Keywords: Distributed memory multicomputer; parallel time stepping; pipelining iteration; time dependent problem

ABSTRAK

Algoritma sela-masa dengan penyelarian ruang umumnya digunakan untuk menyelesaikan persamaan pembezaan separa bersandar masa. Pengiraan pada setiap sela masa dilakukan dengan menggunakan kesemua pemproses yang sedia ada sebelum mara ke sela masa berikutnya. Dalam kes dengan sedikit sahaja komponen ruang yang terlibat dan terdapat banyak pemproses untuk digunakan, algoritma ini akan mengakibatkan kegranulan halus dan pengurangan skalabiliti. Lazimnya satu alternatif dalam kes begini adalah menyelarikan domain masa. Beberapa algoritma penyelarian masa telah dicadangkan sepanjang dua dekad yang lalu. Salah satu daripadanya ialah lelaran penalian paip merentasi sela masa. Walau bagaimanapun dalam kaedah sela masa penalian paip ini, komunikasi di antara sela masa berlaku secara meluas sepanjang proses penalian paip. Ini mengakibatkan penurunan prestasi dalam persekitaran memori teragih yang lazimnya mempunyai latensi mesej yang tinggi. Kami mencadangkan satu algoritma sela-masa penalian paip terubahsuai berdasarkan strategi penalian paip tertangguh dan pengurangan komunikasi bagi memperbaiki masa pelaksanaan keseluruhan dalam persekitaran memori teragih menggunakan MPI. Tujuan kami ialah mengurangkan komunikasi antara sela masa di samping menyediakan maklumat yang mencukupi bagi penumpuan pada sela masa berikutnya. Keputusan berangka menunjukkan algoritma yang ditambahbaik ini adalah lebih pantas berbanding dengan algoritma penalian paip asal dan algoritma sela-masa dengan penyelarian ruang sahaja. Algoritma yang ditambahbaik ini adalah paling berguna bagi masalah bersandar masa berkegranulan halus dengan penyelarian ruang yang terhad.

Kata kunci: Masalah bersandar masa; multikomputer memori teragih; lelaran penalian paip; sela-masa selari

INTRODUCTION

Many physical phenomena in nature are time-dependent and can be modeled by partial differential equations (PDEs). These time-dependent PDEs, when discretised in both spatial and temporal domain using finite difference schemes, will result in a sparse linear system to be solved at each time step. Solution at the desired final time often requires a large number of time steps to compute. Time-marching algorithm with spatial parallelisation alone is commonly used to arrive at the final time required where the linear system at each time step is solved using all processors available before advancing to the next time step, in a sequential manner, due to the fact that time itself is purely sequential. There are various algorithms for solving linear systems utilising the spatial parallelisation at a time step. This approach remains the most effective parallelisation of time dependent PDEs as long as the computation to communication ratio at each time step is

high enough to produce effective speedups. However the scalability of this approach is often lost if the number of processors involved exceeded a threshold limit, which leads to fine granularity. In these cases, adding more processors has little effect on increasing the speedup as more processors would increase the cost of communication compared to computation. Therefore it would be of great value to be able to parallelise the solution process in time dimension as well.

During the last two decades several algorithms have been proposed for solving time dependent PDEs utilising time parallelisation. In such algorithms parts of the solution later in time are computed simultaneously to parts of the solution earlier in time. Examples of such algorithms are pipelined iterations, multigrid waveform relaxation, and Parareal algorithm (Horton & Vandewalle 1995; Lions et al. 2001; Srinivasan & Chandra 2005; Womble 1990). Frantziskonis et al. (2009) combined the Parareal algorithm with the compound wavelet method operating at different spatial and temporal scales. Our paper revisits the pipelining technique to achieve parallelisation in the temporal dimension of time dependent problems. There were also some other methods that used similar pipelining technique but only on spatial parallelisation (Bonomo & Dyksen 1981). Pormann et al. (1998) applied the technique to both spatial and temporal dimensions on Cray T3E. In this paper, we attempt the technique on a relatively high message latency of distributed memory cluster.

We introduce here a parallel time stepping algorithm based on delayed pipeline approach and reduced inter-time step communications for solving two dimensional parabolic PDEs. The work has been motivated by our previous experience of spatial parallelisation in steady-state problems incorporating the Explicit Group (EG) and Explicit Decoupled Group (EDG) methods (Ali & Ng 2006; Ng & Ali 2008). We found that these explicit group methods deteriorate in efficiency when the computation to communication ratio drops to a certain value, and using more processors for the spatial parallelisation would not be beneficial. However, for time dependent problems, we should be able to sustain if not increase the efficiency by redistributing the available processors for use in temporal parallelisation.

This paper documented the proposed algorithm in solving a two-dimensional heat equation, using MPI in a distributed memory multicomputer environment. We employ in our experiments Crank-Nicolson (CN) and Explicit Group (EG) methods as the iterative solvers for the elliptic equations arising at each time step. The paper is organised as follows. In the following section we describe briefly time stepping method for solving time-dependent PDEs and the original pipelined iterations for implementing parallel time stepping. Next we describe the proposed algorithm followed by the numerical experiments and results. Finally, we present the conclusion and some future works.

Time Stepping Algorithm

Sequential Time Stepping

We consider the time-dependent partial differential equations in the form of

$$\begin{aligned} &\frac{\partial u}{\partial t} + Lu = f, \quad x \in \Omega \quad t > t_0 \\ &B(u) = u_b(t), \quad x \in \partial\Omega \quad t > t_0 \\ &u(x,0) = u_0(x), \; x \in \Omega \end{aligned} \tag{1}$$

where L is an elliptic spatial operator, B is the boundary operator, and Ω is a spatial domain with boundary $\partial\Omega$. The equation is discretized in both spatial and temporal dimensions with finite difference schemes. Using a two-level implicit time stepping scheme, we will have at each time step k, an algebraic equation system which is solved by iterative scheme

$$\begin{aligned} &u_k^i = Au_k^{i-1} + P\left(b_k + u_{k-1}\right) \\ &i = 1,2,\ldots \quad k = 1,2,\ldots,T \end{aligned} \tag{2}$$

where A is the iteration matrix, b_k is the discretized form of f, and T is the final time step. Superscript i and subscript k indicate the iteration number and time step respectively. As a simplification for discussion, we denote the iteration scheme (2) as

$$\begin{aligned} &\hat{u}_k^i = \Phi\left(\hat{u}_k^{i-1}, u_{k-1}\right) \\ &i = 1,2,\ldots \quad k = 1,2,\ldots,T \end{aligned} \tag{3}$$

which shows the dependency of a time step intermediate solution of current iteration $\hat{u}_k^i$ on previous intermediate solution $\hat{u}_k^{i-1}$, and also on previous time step converged solution u_{k-1}. The iterative operator Φ can be any iteration solver. We use the classical Crank-Nicolson (CN) method as well as the Explicit Group (EG) method as the iteration solvers in our experiments. Readers are referred to Ng & Ali (2008) for details on the implementation of EG methods.

In a sequential time stepping process of solving (1), execution of each time step k is strictly sequential, much the same as the time in nature. Once a time step is solved, most often by spatial parallelisation, the process advances to the next time step using the just computed solution as $\hat{u}_k^0$ and u_{k-1} in the right hand side of (3). The process continues sequentially until the solution at the desired final time has been computed. If computation in each time step is costly compared to communication incurred, then this approach is very effective due to high computation to communication ratio. However if the spatial domain is limited and many processors are involved, the spatial parallelisation alone will result in fine granularity and the efficiency of the approach will decline. As a result, many researchers turn to time domain as a new dimension for parallelisation.

PARALLEL TIME STEPPING

We mention here one of the earliest methods to exploit time parallelisation was by Womble (1990) and refer to it as pipelined iterations method (Zhu 1994). In this parallel time stepping method, the iterations at time step k>1 starts immediately after the previous time step k-1 has computed the value $\hat{u}_{k-1}^{1}$ at the very first iteration, which is used both as initial guess $\hat{u}_{k}^{0}$ and an approximation for u_{k-1} in the right hand side of (3) for the current time step. Computation at each time step can be carried out by a single processor (i.e. without spatial parallelisation) or group of processors (i.e. with spatial parallelisation). The iterations at time step k will continue simultaneously with the iterations at time step k-1 and all time steps involved proceed like a pipeline, hence the name. Each computed value of $\hat{u}_{k-1}^{i}$ of time step k-1 at iteration i is passed to time step k as a better approximation to u_{k-1} in its own iteration. These inter-time step information transfers will continue until time step k-1 has computed the converged solution u_{k-1}. The efficiency of this parallel time stepping algorithm depends very much on the inter-time step communications which provide better approximations to u_{k-1} after each iteration. However, from the message passing point of view, this can cause a decrease in performance on distributed memory environment which often has high message latency. Every iteration i of each time step k for k=1…T-1 needs to pass the intermediate solution to the next time step. Therefore communication is extensive between time steps during the pipelining process. Since there is always a start-up cost associated with each message, such a large number of inter-time step communications will increase the overall execution time significantly (Srinivasan & Chandra 2005).

DELAYED PIPELINE REDUCED COMMUNICATION TIME STEPPING ALGORITHM

Looking at the high communication costs incurred in pipelined iterations implemented in a distributed memory machines, therefore we seek to reduce the number of communications from a particular time step to the next time step, and meanwhile provide adequate information for the next time step to converge to the correct solution. We refer to our approach as Delayed Pipeline Reduced Communication (DPiRC) parallel time stepping algorithm.

In contrast with the original pipelined iteration, a time step k in DPiRC approach will only send its intermediate solution to next time step k+1 after a number of iterations has elapsed. This is to enable the next time step to receive a more accurate approximation of u_k. Using this delayed intermediate solution, the time step k+1 will then commence its own iterations concurrently, just as in pipelined iterations method. When the iteration at time step k finally has converged, it will then send the converged solution to time step k+1 for computing the correct solution. Hence, we have reduced the number of inter-time step communication to only two, one to start a new time step iteration and the other to ensure the new time step converges to a correct solution. This reduces significantly the overall execution time compared to the original pipelined iteration method in a high message latency environment.

We now define the algorithm more formally. Let us denote u_0 as the known initial value at time step *0*, and $\hat{u}_k^i$ as the intermediate solution of time step k at iteration i, which is computed using an iterative solver as in (3). Let d denotes the predetermined optimal number of iterations to delay the first inter-time step communication in each time step. The pseudocode of the DPiRC algorithm is given in Figure 1. A sample procession of three time steps using the algorithm is given in Figure 2.

NUMERICAL RESULTS AND DISCUSSION

We tested the DPiRC method using the following model problem, a two-dimensional diffusion equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \sin x + \sin y\, e^{-t} - 4$$
$$0 < x, y < 1, \qquad 0 < t < T \tag{4}$$

where the initial and boundary conditions are defined to satisfy the exact solution

$$u(x, y, t) = \sin x + \sin y\, e^{-t} - 4 \tag{5}$$

The experiments were run at a distributed memory computing cluster (Stealth cluster) available at the School of Computer Sciences, Universiti Sains Malaysia. The cluster consists of 1 unit of Sun Fire 280R with 2 processors (900MHz UltraSPARC® III Cu Superscalar SPARC® V9), 2GB RAM, 8MB L2 Cache, and 4 unit of Sun Fire V210 each with 2 processors (1002 Mhz UltraSPARC IIIi), 2GB RAM , 1MB of level 2 cache. The operating system used is Solaris9 (SunOS 2.9) with Sun HPC ClusterTools 5 and Sun MPI 6.0. The cluster was isolated from other network traffic and put under exclusive use during the experiments.

We implemented the DPiRC algorithm using MPI parallel libraries (Gropp et al. 1999) in SPMD master-slave programming approach (Pacheco 1997). We used two iterative solvers in solving (4): Crank-Nicolson (CN) and Explicit Group (EG) methods. For comparison purposes, we modified the same code to run as the sequential time stepping (SeqTS) and the pipelined iterations (PI) versions. Each program was run for 10 times and the best execution time was chosen for performance comparison.

Four performance metrics were measured and reported here, namely

(1) *execution time*, the elapsed time in seconds from synchronised start of data initialisation until end of all solutions
(2) *inter-time step (ITS) communicat*ions, the total count of data transfers from a time step to the next for all k<T (applicable to PI and DPiRC methods only)
(3) *sequential iterations*, the value of $\sum_{k=1}^{T}(m_k - c_k)$ where, m_k is the number of iterations for time step k to converge, and c_k is the iteration at which a time step k receives the previous time step converged solution (note that $c_1 = 0$)

```
Set û_1^0 = u_0
For k=1 to T
    If (k>1) then
        Receive û_{k-1}^d from time step k-1
        Set û_k^0 = û_{k-1}^d
    i = 0
    Do iterative solver
        i = i + 1
        Compute û_k^i = Φ(û_k^{i-1}, u_{k-1})
        If (k<T AND i=d) then
            Send û_k^i to next time step k+1
        If (k>1 AND time step k-1 has converged) then
            Receive u_{k-1} from time step k-1
    Until (û_k^i converged AND time step k-1 has converged)
    If (k<T) then
        Send converged solution u_k = û_k^i to next time step k+1
Next k
```

FIGURE 1. Delayed pipeline reduced communication (DPiRC) time stepping algorithm

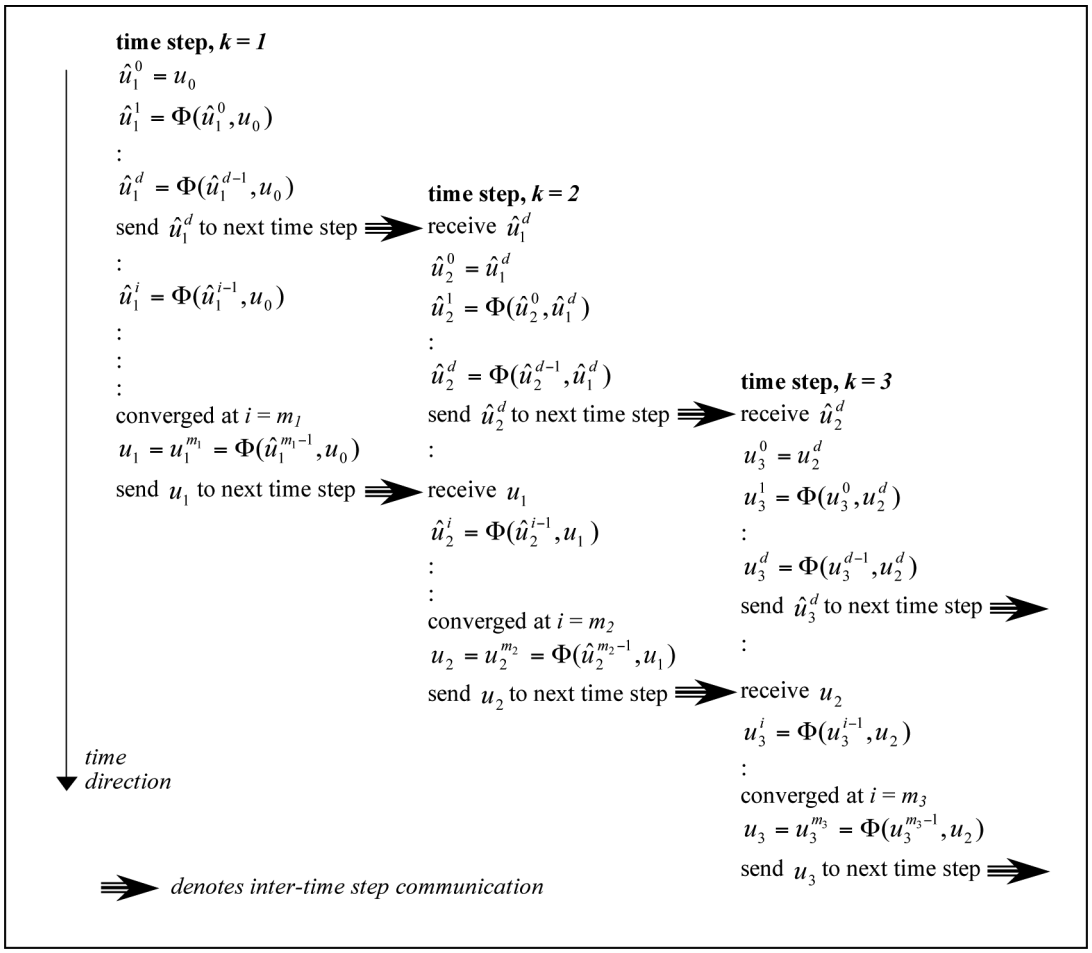

FIGURE 2. Procession of DPiRC iterations

(4) maximum error, *the largest difference between* final computed solutions and the exact solution.

We tested the algorithm with two problem sizes, n=49 and n=97, each using EG and Crank-Nicolson as the iterative solvers. These problem sizes have very limited spatial parallelisation in respect to both iterative solvers, especially n=49. We would expect to gain the benefit of parallelisation by exploiting the temporal dimension using the proposed method. The model problem (4) is solved for final time = 2s with 200 time steps, and termination epsilon value for iterative solver is $1e^{-5}$ in all cases. We use optimal value d=4 for problem size n=49, and d=15 for problem size n=97. For parallel execution of programs, we use p=2, 3 and 4 processors. The results of the four performance metrics for n=49 are summarized in Table 1 and Table 2 , n=97 in Table 3 and Table 4. Table 5 to Table 8 show the corresponding speed-up and efficiency values computed from Table 1 to Table 4 respectively. Figure 3 to Figure 6 are speed-up graphs for the corresponding Table 5 to Table 8.

The results from Table 1 to Table 4 show that the proposed DPiRC algorithm is faster than the pipelined iterations (PI) in solving the model problem. This reduction of execution time in all test cases is expected as the DPiRC algorithm reduces inter-time step communication to merely two counts in all but the final time step. This can be seen from an output excerpts (Figure 7) from a test case where p=3 of Table 1, with only output from the first 20 time steps shown. The output

TABLE 1. Performance of DPiRC compared to SeqTS and PI, using EG as iterative solver with n=49

#Proc	Method	Exec Time	ITS Comms	Seq Iters	Max Error
1	SeqTS	1.1453	-	2760	5.52E-05
2	SeqTS	3.0235	-	2760	5.52E-05
	PI	1.0576	1554	1560	6.54E-05
	DPiRC	0.8034	398	2292	7.84E-05
3	SeqTS	3.1349	-	2760	5.52E-05
	PI	0.9445	2662	1122	7.29E-05
	DPiRC	0.6872	398	1844	7.07E-05
4	SeqTS	3.6826	-	2760	5.52E-05
	PI	0.8926	3518	991	2.31E-05
	DPiRC	0.6245	398	1863	7.42E-05

TABLE 2. Performance of DPiRC compared to SeqTS and PI, using CN as iterative solver with n=49

#Proc	Method	Exec Time	ITS Comms	Seq Iters	Max Error
1	SeqTS	1.4718	-	3689	1.05E-04
2	SeqTS	2.9978	-	3689	1.05E-04
	PI	1.3920	2068	2076	1.26E-04
	DPiRC	1.0358	398	2292	1.41E-04
3	SeqTS	3.6252	-	3689	1.05E-04
	PI	1.2198	3504	1518	1.49E-04
	DPiRC	0.9288	398	2515	1.57E-04
4	SeqTS	4.4336	-	3689	1.05E-04
	PI	1.1637	4763	1416	2.13E-05
	DPiRC	0.8753	398	2516	1.41E-04

TABLE 3. Performance of DPiRC compared to SeqTS and PI, using EG as iterative solver with n=97

#Proc	Method	Exec Time	ITS Comms	Seq Iters	Max Error
1	SeqTS	7.6802	-	4642	1.40E-04
2	SeqTS	8.0813	-	4642	1.40E-04
	PI	8.0059	2596	2606	1.78E-04
	DPiRC	5.5616	398	3310	1.63E-04
3	SeqTS	7.2764	-	4642	1.40E-04
	PI	7.1678	4363	2128	4.36E-05
	DPiRC	4.9730	398	3170	1.67E-04
4	SeqTS	7.6057	-	4642	1.40E-04
	PI	6.6124	6776	2248	1.52E-05
	DPiRC	4.6290	398	3168	1.70E-0

TABLE 4. Performance of DPiRC compared to SeqTS and PI, using CN as iterative solver with n=97

#Proc	Method	Exec Time	ITS Comms	Seq Iters	Max Error
1	SeqTS	8.3075	-	5829	9.31E-05
2	SeqTS	7.6013	-	5829	9.31E-05
	PI	9.3373	3289	3305	6.04E-05
	DPiRC	5.8411	398	3971	2.21E-04
3	SeqTS	7.8344	-	5829	9.31E-05
	PI	11.6849	7691	3831	4.71E-05
	DPiRC	6.0727	398	4470	1.89E-04
4	SeqTS	8.8114	-	5829	9.31E-05
	PI	13.1687	12358	4051	3.85E-05
	DPiRC	6.2145	398	4679	1.95E-04

TABLE 5. Speedup and efficiency of DPiRC compared to SeqTS and PI using EG as iterative solver with n=49

#Proc	Method	Exec Time	Speed up	Efficiency
1	SeqTS	1.1453	1.0000	100.0%
2	SeqTS	3.0235	0.3788	18.9%
	PI	1.0576	1.0829	54.1%
	DPiRC	0.8034	1.4256	71.3%
3	SeqTS	3.1349	0.3653	12.2%
	PI	0.9445	1.2126	40.4%
	DPiRC	0.6872	1.6666	55.6%
4	SeqTS	3.6826	0.3110	7.8%
	PI	0.8926	1.2831	32.1%
	DPiRC	0.6245	1.8339	45.8%

TABLE 6. Speedup and efficiency of DPiRC compared to SeqTS and PI using CN as iterative solver with n=49

#Proc	Method	Exec Time	Speed up	Efficiency
1	SeqTS	1.4718	1.0000	100.0%
2	SeqTS	2.9978	0.4910	24.5%
	PI	1.3920	1.0573	52.9%
	DPiRC	1.0358	1.4209	71.0%
3	SeqTS	3.6252	0.4060	13.5%
	PI	1.2198	1.2066	40.2%
	DPiRC	0.9288	1.5846	52.8%
4	SeqTS	4.4336	0.3320	8.3%
	PI	1.1637	1.2648	31.6%
	DPiRC	0.8753	1.6814	42.0%

TABLE 7. Speedup and efficiency of DPiRC compared to SeqTS and PI using EG as iterative solver with n=97

#Proc	Method	Exec Time	Speed up	Efficiency
1	SeqTS	7.6802	1.0000	100.0%
2	SeqTS	8.0813	0.9504	47.5%
	PI	8.0059	0.9593	48.0%
	DPiRC	5.5616	1.3809	69.0%
3	SeqTS	7.2764	1.0555	35.2%
	PI	7.1678	1.0715	35.7%
	DPiRC	4.9730	1.5444	51.5%
4	SeqTS	7.6057	1.0098	25.2%
	PI	6.6124	1.1615	29.0%
	DPiRC	4.6290	1.6591	41.5%

TABLE 8. Speedup and efficiency of DPiRC compared to SeqTS and PI using CN as iterative solver with n=97

#Proc	Method	Exec Time	Speed up	Efficiency
1	SeqTS	8.3075	1.0000	100.0%
2	SeqTS	7.6013	1.0929	54.6%
	PI	9.3373	0.8897	44.5%
	DPiRC	5.8411	1.4223	71.1%
3	SeqTS	7.8344	1.0604	35.3%
	PI	11.6849	0.7110	23.7%
	DPiRC	6.0727	1.3680	45.6%
4	SeqTS	8.8114	0.9428	23.6%
	PI	13.1687	0.6308	15.8%
	DPiRC	6.2145	1.3368	33.4%

shows that for pipelined iterations algorithm, the inter-time step communications (# ITS comms) occur for as many as the number of iterations in each time step, in contrast with DPiRC algorithm which incurs only two communications at each time step before the final time. The price of reducing the amount of information transfer is the increase of the number of sequential iterations (# Seq Iters) in each time step. However, the reduction of communication cost is more dominant than the increase of computation cost, hence decreasing the total execution time.

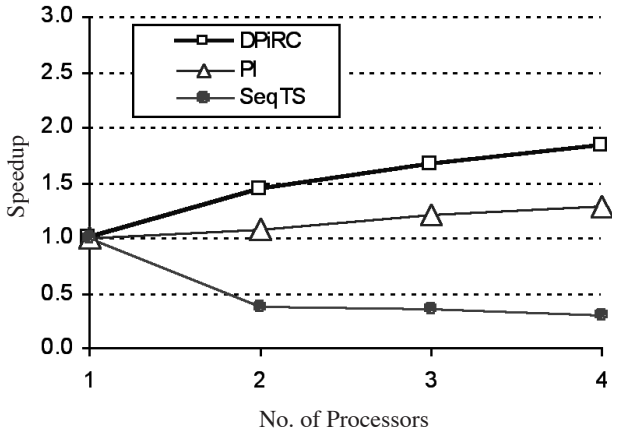

FIGURE 3. Speedup Performance of DPiRC compared to SeqTS and PI, using EG as iterative solver for n=49

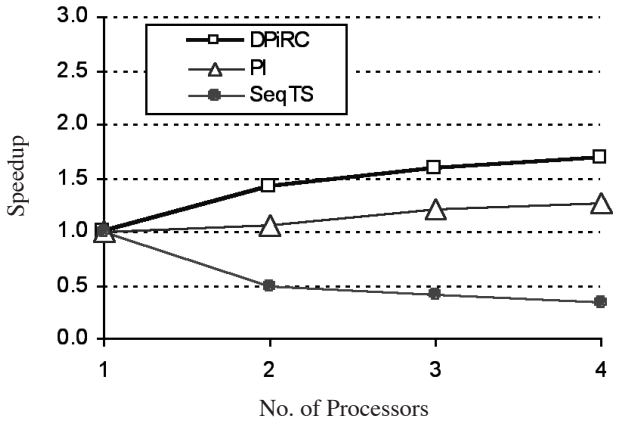

FIGURE 4. Speedup Performance of DPiRC compared to SeqTS and PI, using CN as iterative solver for n=49

The decrease of execution time is most apparent in problem size where spatial parallelisation is limited (n=49) as shown in Table 1 and Table 2. In this fine granularity problem, using more than one processor for spatial parallelisation (SeqTS with p>1) causes the execution time to increase due to the high communication to computation ratio. By using the proposed algorithm, we have demonstrated that we still could achieve some degree of efficiency when spatial parallelisation failed to do so. As shown in Table 5 and Table 6, DPiRC algorithm is able to maintain higher efficiency ranging from 45.8% to 71.3% and 42.0% to 71.1% for the respective test cases when spatial parallelisation alone caused the efficiency to drop to as low as 7.8% and 8.3%. Similar results pattern can also be seen in Table 7 and Table 8 for n=97. These results demonstrated the feasibility and possible benefit of parallelisation in time dimension when parallelisation in space dimension alone failed to reduce the execution time. However, the original pipelined iterations (PI) algorithm has low efficiency in distributed memory computing environment due to the algorithm's nature of extensive communication labor. By reducing the inter-time step

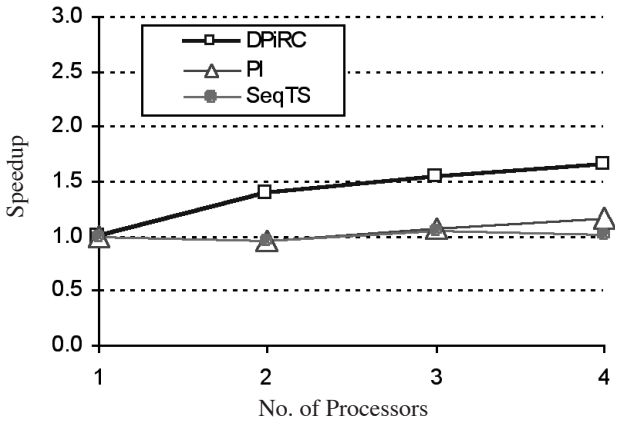

FIGURE 5. Speedup Performance of DPiRC compared to SeqTS and PI, using EG as iterative solver for n=97

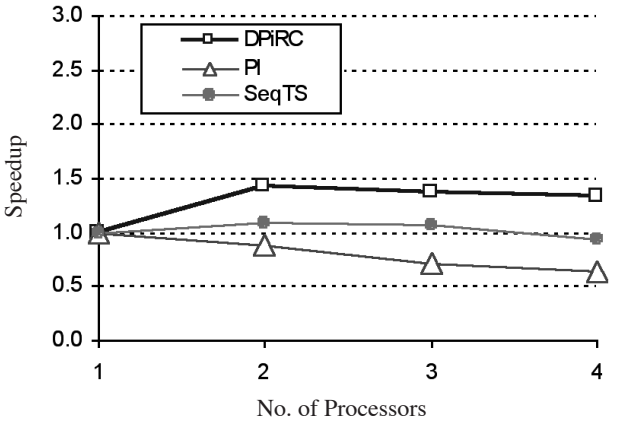

FIGURE 6. Speedup Performance of DPiRC compared to SeqTS and PI, using CN as iterative solver for n=97

Method: SeqTS

Time step *k*	=	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Total
# Seq Iters	=	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	320

(no ITS communications for SeqTS)

Method: PI

Time step *k*	=	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Total
# iteration	=	16	22	27	19	24	19	19	21	20	19	21	24	17	21	19	23	24	18	23	24	420
# ITS comms	**=**	**16**	**22**	**13**	**19**	**14**	**13**	**16**	**15**	**13**	**16**	**19**	**11**	**15**	**14**	**18**	**19**	**12**	**18**	**19**	**14**	**316**
# Seq Iters	=	16	7	6	7	6	6	7	6	6	7	6	6	7	7	6	6	6	7	6	6	137

Method: DPiRC

Time step *k*	=	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Total
# iteration	=	16	26	34	26	28	28	25	24	24	22	22	23	22	22	22	20	21	23	21	21	470
# ITS comms	**=**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**40**	
# Seq Iters	=	16	14	14	14	13	13	12	12	11	11	11	11	11	11	10	10	11	11	10	11	237

FIGURE 7. Communication and sequential iteration counts for test cases of Table 1 for the first 20 time steps

communications in the proposed algorithm we achieved better efficiency than the pipelined iterations algorithm, as shown in Table 5 to Table 8.

CONCLUSION

In this paper, an improved pipelined algorithm has been proposed for parallelising time stepping process in solving time dependent problems and some preliminary results has been reported. We have shown that the algorithm is most beneficial for problems with limited spatial parallelisation or when spatial parallelisation alone has reached the efficiency limit. However, we see time parallelisation as an addition to spatial parallelisation rather as a competitor. We can and should combine both. If a given problem can be parallelized efficiently through spatial decomposition then a group of processors can be assigned to a time step. More study will be carried out to further investigate and to derive a performance analysis model of the algorithm. This includes the determination of the optimal or near optimal pipeline delay value, *d*.

REFERENCES

Ali, N.H.M. & Ng, K.F. 2006. Explicit group PDE solvers in an MPI environment. *International Conference on Mathematical Modelling and Computation*, June 5-8, Universiti Brunei Darussalam.

Bonomo, J.P. & Dyksen, W.R. 1981. Pipelined iterative methods for shared memory machines. *Technical Report 688*, Computer Science Department, Purdue University.

Frantziskonis, G., Muralidharan, K., Deymier, P., Simunovic, S., Nukala, P. & Pannala S. 2009. Time-parallel multiscale/ multiphysics framework. *J. Comput. Phys.* 228: 8085-8092.

Gropp, W., Lusk, E. & Skjellum, A. 1999. *Using MPI: Portable parallel programming with the message-passing interface*. Cambridge, MA: MIT Press.

Horton, G. & Vandewalle, S. 1995. A space-time multigrid method for parabolic PDEs. *SIAM J. Sci. Computing* 16(4): 848-864.

Lions, J-L., Maday, Y. & Turinice, G. 2001. A "parareal" in time discretization of PDE's. *C. R. Acad. Sci. Paris Ser. I Math.* 332: 661-668.

Ng, K.F. & Ali, N.H.M. 2008. Performance analysis of explicit group parallel algorithms for distributed memory multicomputer. *Parallel Computing* 34(6-8): 427-440.

Pacheco, P.S. 1997. *Parallel Programming with MPI*. San Francisco: Morgan Kaufmann Publishers.

Pormann, J.B., Board, J.A. Jr. & Rose, D.J. 1998. The implicit pipeline method. *Proceedings of the 12th. International Parallel Processing Symposium on International Parallel Processing Symposium*. IEEE Computer Society, Washington, DC 721-725.

Srinivasan, A. & Chandra, N. 2005. Latency tolerance through parallelization of time in scientific applications. *Parallel Computing* 31(7): 777-796.

Womble, D.E. 1990. A time stepping algorithm for parallel computers. *SIAM J. Sci. Computing* 11(5): 824-837.

Zhu, J. 1994. *Solving partial differential equations on parallel computers*. Singapore: World Scientific Publishing.

School of Mathematical Sciences
Universiti Sains Malaysia
11800 Penang
Malaysia

*Corresponding author; email: ngkokfu@gmail.com

Received: 19 May 2009
Accepted: 1 June 2010